AD-A261 441

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FINAL/01 NOV 90 TO 30 APR 92

INTELLIGENT, REAL-TIME PROBLEM SOLVING (U)

5. FUNDING NUMBERS

5956/00/DARPA
AFOSR-91-0067

6. AUTHOR(S)

Professor Paul Cohen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Massachusetts
Computer Science Department
Amherst, MA 01003

8. PERFORMING ORGANIZATION REPORT NUMBER

AFOSR-TR- 93 0063

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFOSR-91-0067

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Progress was made in the IRTPS initiative in five areas: resolving the relationship between deliberative and reactive planning with a framework for their effective combination; modeling the real-time performance of the Phoenix planner through the application of a powerful statistical technique - path analysis- that can be used to build causal models from observed behavior; developing and applying design-to-time scheduling to design a solution that uses all available resources to maximize solution quality within available time; extending previous work on the real-time monitoring and control structure we call envelopes; and finally, in a new body of research started in Phase III, delineating a taxonomy of monitoring problems found in real-time domains, and developing optimal monitoring strategies to address these problems.

14. SUBJECT TERMS

15. NUMBER OF PAGES
19

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR(SAME AS REPORT)

Intelligent, Real-time Problem Solving

Phase III Final Report

Paul R. Cohen, *Principal Investigator*
Victor R. Lesser, *Principal Investigator*
David M. Hart, *EKSL Lab Manager*

Experimental Knowledge Systems Laboratory
Cooperative Distributed Problem-Solving Laboratory
Computer Science Department
University of Massachusetts
Amherst, MA 01003

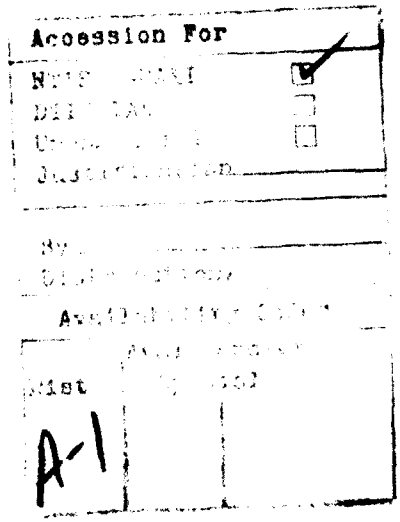
Final Report for the
Air Force Office of Scientific Research
Contract No. AFOSR-91-0067
Dr. Abraham Waksman, *Program Manager*
Contract period: 11/1/90 – 4/30/92

Abstract

We report progress made during Phase III of the IRTPS initiative in five areas: resolving the relationship between *deliberative* and *reactive* planning with a framework for their effective combination; modeling the real-time performance of the Phoenix planner through the application of a powerful statistical technique – *path analysis* – that can be used to build causal models from observed behavior; developing and applying *design-to-time* scheduling to design a solution that uses all available resources to maximize solution quality within available time; extending previous work on the real-time monitoring and control structure we call *envelopes*; and finally, in a new body of research started in Phase III, delineating a taxonomy of monitoring problems found in real-time domains, and developing optimal monitoring strategies to address these problems.

1 Numerical Productivity Measures

Refereed papers published: 5
Refereed papers submitted: 4
Unrefereed reports and articles: 11
Books or parts thereof published: 2
Invited presentations: 9
Contributed presentations: 3
Tutorials: 2
Honors, including conference committees: 6
Graduate students supported at least 25% time: 6



2 Executive Summary

We report progress made during Phase III of the IRTPS initiative in five areas:

Resolving the Deliberative/Reactive Dilemma. One of the central dilemmas in AI in the past several years, particularly for those of us interested in real-time performance, has been understanding the relationship between *deliberative* and *reactive* planning. Our work in Phase III has led to a dissertation investigation that will use representations of *stochastic processes* and a search algorithm to produce distributions of future states. These distributions can then be used to aid in planning how to solve the problem. Predicting the future, or *temporal projection*, is very computationally expensive, and will is not warranted in all environments and for all tasks. The central claim of this work is that the intrinsic uncertainty of an agent's environment, the measurement error of its sensors, and the inexactness of its effectors collectively *bound the depth* of temporal lookahead the agent can do.

***Real Time Knob* Baselining Experiment and Path Analysis.** Part of our proposed effort for Phase III was to baseline the real-time performance of the Phoenix Fireboss in order to design real-time scheduling algorithms for its cognitive activities. We undertook the experiment described in Hart & Cohen (1992) to measure how changes in the Fireboss's thinking speed affected its real-time performance. To analyze the results we used a statistical modeling technique called *path analysis* that we feel holds great promise as a method for building causal models from empirical observations of AI planner behavior.

Design-to-Time Scheduling. *Design-to-time* is an approach to problem-solving in resource-constrained domains that involves designing a solution to a problem that uses all available resources to maximize the solution quality within the available time. During Phase III we have worked to define the design-to-time approach in detail, contrasting it to the anytime algorithm approach, and to develop a heuristic algorithm for design-to-time real-time scheduling.

E .velopes. During Phase III we extended our work on the real-time monitoring and control structure we call *envelopes*. We showed previously that envelopes could be applied in Phoenix to compare the actual progress of plans to the expected progress on which the plans were based. Such a comparison can be used to predict plan failure in advance, thus allowing the planning system a head start in responding to the failure. During Phase III we showed that good performance can be achieved by hand-constructed slack time envelopes, and we presented a probabilistic model of progress, from which we derived a method for automatically constructing slack time envelopes that balances the benefits of *early warning* that a plan is failing against the costs of *false positives* (erroneous predictions that a plan is failing caused by uncertainty in our predictions or serendipitous events).

Timing is Everything. The value of actions depend on their timing. Actions that interact with processes are more or less effective depending on when they occur. We are familiar with the idea of a window of opportunity, but it isn't always clear how to recognize such a window before it closes. Sometimes it is advantageous to monitor processes to detect windows. We have identified a number of timing problems; some require monitoring and some don't. We believe that the large and somewhat bewildering range of timing problems might be described by relatively few features. These features can be composed to form a *taxonomy of monitoring problems.* Having established a taxonomy, we set about tackling some of the monitoring problems in it, such as:

- a Phoenix monitoring problem (how often to look for new fires)
- monitoring to predict when a task will finish
- monitoring to predict whether a task will meet a deadline

In the last case, we have shown that given an explicit model of task execution time, execution cost, and payoff for meeting a deadline, an *optimal decision rule* for monitoring each task can be constructed using *stochastic dynamic programming.* Furthermore, if a model is not available, the same rule can be *learned* using *temporal difference methods.* These results are significant because of the growing importance of such decision rules, or monitoring strategies, in real-time computing.

Research in the first four areas mentioned above has proceeded for several years and is beginning to mature, as evidenced by the papers that have been published in these areas (see Section 4, Publications, Technical Reports, Presentations and Honors). The work described in the fifth area, Timing is Everything, began about a year ago and is just now starting to produce publishable results (see Section 4 under Refereed Papers Submitted). This is an exciting new research direction growing out of the work we have done with envelopes and with the modeling of AI planner behavior[1]. We think this work has important applications in real-time computing – in operating systems as well as AI.

Much of the work we have done for the IRTPS initiative is being transferred to the DARPA/Rome Labs Planning Initiative (PI). This work, discussed in Section 6 (Interactions) includes:

- the creation of a *testbed environment* (our Phase I work). The PI's Common Prototyping Environment will have many of the testbed features we built into Phoenix and that we are now building into a transportation planning simulation.
- our ongoing work with envelopes and monitoring, which is directly applicable to the design of *pathology demons* for the transportation planning problem. Pathology demons detect typical pathologies that arise during the execution of transportation plans and to help the user *steer the plan around the pathologies.*
- the use of path analysis to build causal models of program behavior. As part of a new contract in the PI, we will be building an experiment module that automatically generates causal models from data sets to *aid developers in the design and evaluation of large-scale AI planning systems.*

---

[1] The Modeling, Analysis and Design approach to AI research (Cohen 1991b) was developed as part of Paul Cohen's ONR/AFOSR sponsored sabbatical work in 1989-90.

3 Detailed Statement of Work and Status of Research Effort

3.1 Introduction

During Phase I of the IRTPS initiative we developed the Phoenix system into a full simulation testbed for the conduct of real-time research in complex planning environment. Though there was a lag between the end of Phase I and the beginning of Phase III, we began our program of real-time research even before Phase I was completed. Some of the work done at this time is worth mentioning here as introduction to the reported results of Phase III that follow in Sections 3.2 – 3.7.

There were two broad themes to this work. The first is the development of envelopes for tracking the progress of plans and giving early warning of failure. This development spawned a corollary interest in the frequency of monitoring, since the Phoenix planner must use cognitive resources to monitor progress that, given the real-time nature of the problem, must be shared with the resources used for planning, communication, and any other necessary cognitive activities. The question then becomes, what is the optimal monitoring frequency that 1) looks at the state of the plan often enough to give adequate warning of failure while 2) diverting only minimally the cognitive resources needed to do the agent's other work? Envelopes and monitoring frequency are investigated in two papers from this period:

Envelopes as a Vehicle for Improving the Efficiency of Plan Execution (Hart, Anderson & Cohen, 1990). Envelopes are structures which capture expectations of the progress of a plan. By comparing expected progress with actual progress, envelopes can notify the planner when the plan violates those expectations. The planner then has the opportunity to modify the plan to increase its efficiency given the unexpected progress. This paper presents a specific example of the construction and use of an envelope, followed by a discussion of the general utility of envelopes for improving the efficiency of plan execution.

Monitoring Interval (Anderson & Hart, 1990). We attempt to determine bounds on the rate at which an agent must sample and react to the environment, given parameters describing the average and maximum rates of change of that environment. We determine the time it takes to encircle a fire with fireline, in the average and worst case, given the average and maximum rate of change of the spread rate of the fire. Other necessary parameters are the initial perimeter of the fire, the cutting rate of a bulldozer, and the "advantage" of the fire-fighting forces, that is, the amount by which the number of bulldozers sent to the fire exceeds the minimum necessary.

The issues addressed in these papers led to our Phase III research into the relationship between deliberative and reactive planning (Section 3.2), the design of task schedulers that give the best solution in the time available (Section 3.4), and the tradeoff between the replanning head-start given by early warning of plan failure and the cost of replanning needlessly incurred when the prediction of failure is wrong (Section 3.5).

The second main theme of our research in the period between Phase I and Phase III was the development of a formal methodology for the conduct of AI research (this work was supported by a small grant for ONR/AFOSR). The design of AI systems is typically justified informally. For example, one might say, "The planner is designed to be reactive because the environment changes rapidly in unexpected ways." We believe this style of justification is too informal to support (a) demonstrations of the necessity of a design, (b) evaluation of the design, (c) generalization of the design to other tasks and environments, (d) communication of the design to other researchers, and (e) comparisons between designs. Through our current research program we seek to demonstrate that achieving these goals is a natural consequence of basing designs on formal models of the interactions between agents and their environments.

The methodology we have developed for this purpose we call Modeling, Analysis and Design (MAD). In Cohen (1991b) we surveyed the papers presented at AAAI-90 and concluded that there was a real need for some such formalisation of methodological approach in AI. Also in this paper we presented the MAD methodology in some detail. Since that time we have refined the MAD approach in the course of applying it to all of the research conducted in our lab. This has led to (among other things) the "discovery" that we could apply a little-used statistical technique called path analysis to build causal models that would explain why our programs behave the way they do, thus allowing us to make substantiable claims about performance and informed decisions about what changes to make to enhance performance (see Section 3.3).

Both of these themes came together in the identification and (beginning) explorations of timing problems found in Section 3.6, Timing is Everything. A survey of the literature (Hansen & Cohen 1992c) suggests that the general problem of monitoring has not been fully addressed by the real-time computing community (AI and otherwise). Our early investigations into the appropriate monitoring interval for a Phoenix problem, along with our adherence to the modeling, analysis and design approach, prompted us to tackle this important problem in a structured way, defining a taxonomy of significant monitoring problems and then constructing optimal strategies for addressing as many of these problems as we can.

3.2 Resolving the Deliberative/Reactive Dilemma

One of the central dilemmas in AI in the past several years, particularly for those of us interested in real-time performance, has been deliberative planning versus reactive planning. The former is what AI has called planning since the time of STRIPS, while the latter is a reaction to the computational complexity of formal planning and emphasizes algorithms that simply react to the current situation. From the beginning of the Phoenix project, we saw a need for both kinds of capability: the foresight of a deliberative planner in order to coordinate the resources of multiple firefighting field agents, and the speed of a reactive planner in order to survive sudden changes in the direction and location of the fire. Consequently, we designed what we have called a "two-loop" architecture where the outer sense-act loop (the Phoenix *cognitive*

scheduler) is slow enough to accommodate deliberative planning while the inner sense-act loop (a Phoenix agent's *reflexes*) is quick enough to react to a changing environment (Cohen, et al., 1989).

In our initial investigations of this design's efficacy, we derived a simple mathematical model of the interaction of two tasks running concurrently on a uniprocessor (Anderson, Hart & Cohen 1991). Specifically, we modeled the rate at which two periodic tasks would interrupt each other. The predictions of this model were supported by simulation experiments. From the model, we argued that – contrary to our expectations – placing reflexive control and cognitive control on different processors is not justified by the different time-scales over which they act. It *is*, however, justified by the extent to which the higher priority task dominates the uniprocessor.

The research question we've been investigating since then is essentially: *when should the agent deliberate and when should it react*? What are the dimensions of that decision? We believe the answer rests on the following principles and criteria:

- Deliberation, especially "Forward Search," is computationally expensive
- Deliberation is sometimes useful or even necessary
- Its use is dictated by:
  - Computational Cost vs. Execution Benefit
  - Abstraction Level
  - Uncertainty
    - * Intrinsic unpredictability of the world
    - * How must precision does the agent demand?

To attack the question of cost and benefit, we implemented an artificial world for path planning (navigation), consisting of linear obstacles of random length and orientation. These obstacles, called "chaff" (hence the world was called "Chaffworld") can be avoided more efficiently by planning a path around them (*deliberating*) rather than bumping into them (*reacting*), but this efficiency is bought at the cost of that planning.

Chaffworld also partially addresses the question of abstraction level. The abstraction level is the primary factor in assessing the cost of deliberation in the form of search: Deliberation will always cost less when done at a higher abstraction level. However, the disadvantage of higher abstraction levels is greater error in the planning, hence less benefit. In Chaffworld, the agent planned using a "map" that included only long chaff (longer than some threshold).

Finally, we are addressing the question of the effect of uncertainty on the tradeoff between deliberating and reacting by designing an agent that is sensitive to the uncertainty of its environment and deliberates until the accumulated uncertainty makes further deliberation futile. We believe that such an agent will act effectively in both highly uncertain (high entropy) and less uncertain (low entropy) worlds. Furthermore, we believe that the agent will naturally act in a deliberative manner in low-entropy worlds and reactively in high-entropy worlds.

Essentially, we claim that the design of an agent must reflect the task it undertakes and the environment it experiences. In some environments, the agent will be reac-

tive, while in others it will be deliberative. In environments that exhibit qualities of both reactive and deliberative environments, the agent design may well be a "two-loop" design. We can design the right kind of agent for its environment by knowing the cost/benefit tradeoff in the task, the level of abstraction of the representation used in deliberation, and the entropy of the environment.

Thesis Proposal. The work described above has led to the formulation of a thesis proposal to tackle the deliberative/reactive question: When an agent attempts to solve an ongoing problem (one whose state changes with time, such as a fire that continues burning while firefighters work), the agent must address the future state of the problem, not simply its current state. Therefore, the agent must project what that future state might be, by using some model of the problem process. In general, that process might be non-deterministic, and so the problem would be a *stochastic* process. This dissertation (Anderson 1992) that will investigate using representations of stochastic processes and a search algorithm to produce distributions of future states. These distributions can then be used to aid in planning how to solve the problem.

Predicting the future like this, or *temporal projection*, is very computationally expensive, and will not be warranted in all environments and for all tasks. The central claim of the dissertation is that the intrinsic uncertainty of an agent's environment, the measurement error of its sensors, and the inexactness of its effectors collectively bound the depth of temporal lookahead the agent can do. All of these uncertainty factors interact strongly with the resolution of the agent's model of the environment, and so the level of resolution is another important factor. Other factors that affect the depth of lookahead are the speed of the agent's brain, which determines the cost of lookahead, and the nature of the task, which determines the benefit. The resulting depth of temporal lookahead determines whether the agent's behavior is termed "reactive" (shortsighted) or "deliberative" (farsighted).

3.3 *Real Time Knob* Baselining Experiment and Path Analysis

Part of our proposed effort for Phase III was to baseline the real-time performance of the Phoenix Fireboss in order to design real-time scheduling algorithms for its cognitive activities. We undertook the experiment described in Hart & Cohen (1992) to measure how changes in the Fireboss's thinking speed affected its real-time performance. To analyze the results we used a statistical modeling technique called *path analysis* that we feel holds great promise as a method for building causal models from empirical observations of AI planner behavior.

It is difficult to predict or even explain the behavior of any but the simplest AI programs. A program will solve one problem readily, but make a complete hash of an apparently similar problem. For example, our Phoenix planner, which fights simulated forest fires, will contain one fire in a matter of hours but fail to contain another under very similar conditions. We therefore hesitate to claim that the Phoenix planner "works." The claim would not be very informative, anyway: we would much rather be able to predict and explain Phoenix's behavior in a wide range of conditions (Cohen 1991b). In Hart & Cohen (1992) we describe an experiment with Phoenix in

which we uncover factors that affect the planner's behavior and test predictions about the planner's robustness against variations in some factors. We also introduce a technique—path analysis—for constructing and testing causal explanations of the planner's behavior. Our results are specific to the Phoenix planner and will not necessarily generalize to other planners or environments, but our techniques are general and should enable others to derive comparable results for themselves.

We designed an experiment with two purposes. A *confirmatory* purpose was to test predictions that the planner's performance is sensitive to some environmental conditions but not others. In particular, we expected performance to degrade when we change a fundamental relationship between the planner and its environment—the amount of time the planner is allowed to think relative to the rate at which the environment changes—and not be sensitive to common dynamics in the environment such as weather, and particularly, wind speed. We tested two specific predictions: 1) that performance would not degrade or would degrade gracefully as wind speed increased; and 2) that the planner would not be robust to changes in the Fireboss's thinking speed due to a bottleneck problem described below. An *exploratory* purpose of the experiment was to identify the factors in the Fireboss architecture and Phoenix environment that most affected the planner's behavior, leading to a causal model of the time required to put out a fire.

The Fireboss must select plans, instantiate them, dispatch agents and monitor their progress, and respond to plan failures as the fire burns. The rate at which the Fireboss thinks is determined by a parameter called the *Real Time Knob*. By adjusting the Real Time Knob we allow more or less simulation time to elapse per unit CPU time, effectively adjusting the speed at which the Fireboss thinks relative to the rate at which the environment changes.

The Fireboss services bulldozer requests for assignments, providing each bulldozer with a task directive for each new fireline segment it builds. The Fireboss can become a bottleneck when the arrival rate of bulldozer task requests is high or when its thinking speed is slowed by adjusting the Real Time Knob. This bottleneck sometimes causes the overall digging rate to fall below that required to complete the fireline polygon before the fire reaches it, which causes replanning. In the worst case, a Fireboss bottleneck can cause a thrashing effect in which plan failures occur repeatedly because the Fireboss can't assign bulldozers during replanning fast enough to keep the overall digging rate at effective levels. We designed our experiment to explore the effects of this bottleneck on system performance and to confirm our prediction that performance would vary in proportion to the manipulation of thinking speed. Because the current design of the Fireboss is not sensitive to changes in thinking speed, we expect it to take longer to fight fires and to fail more often to contain them as thinking speed slows.

In contrast, we expect Phoenix to be able to fight fires at different wind speeds. It might take longer and sacrifice more area burned at high wind speeds, but we expect

this effect to be proportional as wind speed increases and we expect Phoenix to succeed equally often at a range of wind speeds, since it was designed to do so.

In Hart & Cohen we show that performance did indeed degrade as we systematically slowed Fireboss thinking speed. Interestingly, this degradation was not linear (with respect to the time required to contain the fire). We tried using regression to model the factors that determine this nonlinear relationship, but found that while we could derive a predictive model, such a regression model doesn't allow us to *explain* the inter-related causal influences among the factors. We were able to apply path analysis (Li 1975; Asher 1983) to build a model that is both predictive *and* explanatory, and which tells us (among other things) how Phoenix performance will be affected by changes in the amount of thinking time available to the Fireboss.

3.4 Design-to-Time Scheduling

Design-to-time, a generalization of what we have previously called *approximate processing* (Lesser, Pavlin & Durfee, 1988), is an approach to problem-solving in resource-constrained domains where: multiple solution methods are available for tasks, those solution methods make tradeoffs in solution quality versus time, and satisficing solutions are acceptable. Design-to-time involves designing a solution to a problem that uses all available resources to maximize the solution quality within the available time. In Garvey & Lesser (1992) we define the design-to-time approach in detail, contrasting it to the anytime algorithm approach, and presents a heuristic algorithm for design-to-time real-time scheduling. The methodology is known as design-to-time because it advocates the use of all available time to generate the best solutions possible. It is a problem-solving method of the type described by D'Ambrosio (1989) as those which "given a time bound, dynamically construct and execute a problem solving procedure which will (probably) produce a reasonable answer within (approximately) the time available."[2]

This form of problem-solving is related to – but distinct from – the use of *anytime algorithms*. Boddy and Dean (1989) describe anytime algorithms as interruptible procedures that always have a result available and that are expected to produce better results as they are given additional time. Russell and Zilberstein (1991) make a distinction between two kinds of anytime algorithms: *interruptible* algorithms, which can be interrupted at any time and always produce a result, and *contract* algorithms, which must be given a time allocation in advance, and produce better results with increased time allocations, but may produce no useful results if interrupted before their allocated time. Design-to-time differs from contract anytime algorithms in that we have a predefined set of solution methods with discrete duration and quality values. Contract anytime algorithms can generate a solution method of any duration; that duration just has to be specified before task execution begins.

---

[2] It appears that Bonissone and Halverson (1990) were the first to use the term "design-to-time" to refer to systems of the form described by D'Ambrosio.

In Garvey & Lesser (1992), a blackboard architecture that implements the design-to-time approach is discussed and an example problem and solution from the Distributed Vehicle Monitoring Testbed (DVMT) is described in detail. Experimental results, generated using a simulation, show the effects of various parameters on scheduler performance.

This work is part of Alan Garvey's Ph.D. research.

3.5 Envelopes: The Tradeoff Between Early Warning and False Positives

During Phase III we extended our work on the real-time monitoring and control structure we call *envelopes*. Work we showed previously that envelopes could be applied in Phoenix to compare the actual progress of plans to the expected progress on which the plans were based (Hart, Anderson & Cohen, 1990). Such a comparison can be used to predict plan failure in advance, thus allowing the planning system a head start in responding to the failure.

Underlying the judgment that a plan will not succeed is a fundamental tradeoff between the cost of an incorrect decision and the cost of evidence that might improve the decision. For concreteness, let's say a plan succeeds if a vehicle arrives at its destination by a deadline, and fails otherwise. At any point in a plan we can correctly or incorrectly predict that the plan will succeed or fail. If we predict early in the plan that it will fail, and it eventually fails, then we have a hit, but if the plan eventually succeeds we have a false positive. False positives might be expensive if they lead to replanning. In general, the false positive rate decreases over time (e.g., very few predictions made immediately before the deadline will be false positives) but the reduction in false positives must be balanced against the cost of waiting to detect failures. Ideally, we want to accurately predict failures as early as possible; in practice, we can have accuracy or early warnings but not both.

The false positive rate for a decision rule that at time t predicts failure will generally decrease as t increases. In our Phase III work, detailed in Cohen, St. Amant & Hart (1992), we analyze this tradeoff in several ways. First, we describe a very simple decision rule, called a *slack time envelope*, that we have used for years in the Phoenix planner. Then, using empirical data from Phoenix, we evaluate the false positive rate for envelopes and show that envelopes can maintain good performance throughout a plan. An infinite number of slack time envelopes can be constructed for any plan, and the first analysis in the paper depends on "good" envelopes constructed by hand. To be generally useful, envelopes should be constructed automatically. This requires a formal model of the tradeoff between when a failure is predicted (earlier is better) and the false positive rate of the prediction, shown next in the paper. Finally we show how the conditional probability of a plan failure given the state of the plan can be used to construct "warning" envelopes.

Although we rely heavily on slack time envelopes in the Phoenix planner, we have always constructed them by heuristic criteria, and we did not know how to evaluate their performance. In this work we showed that good performance can be achieved by

hand-constructed slack time envelopes, and we presented a probabilistic model of progress, from which we derived a method for automatically constructing slack time envelopes that balances the benefits of early warnings against the costs of false positives. Our contribution has been to cast the problem in probabilistic terms and to develop a framework for evaluation. We are presently extending our work to other models of progress and different, more complex domains.

3.6 Timing is Everything

The value of actions depend on their timing. Actions that interact with processes are more or less effective depending on when they occur. We are familiar with the idea of a window of opportunity, but it isn't always clear how to recognize such a window before it closes. Sometimes it is advantageous to monitor processes to detect windows. In (Cohen 1991a) we describe several timing problems; some require monitoring and some don't. We consider two monitoring strategies, the periodic strategy for monitoring for fires, and the cupcake strategy. We have proved the optimality of the former, and "mature" humans engage in the latter, although it isn't necessarily optimal. We also describe one case in which the cost functions for two processes are combined. Cohen (1991a) ends by suggesting that the large and somewhat bewildering range of timing problems might be described by relatively few features.

These features are listed below in Section 3.6.1, and can be composed to form what we think is a preliminary *taxonomy of monitoring problems.* Having established this taxonomy, we set about tackling some of the monitoring problems. These efforts are described in subsections 3.6.2 – 3.6.7.

We can differentiate monitoring situations by the behavior of processes:

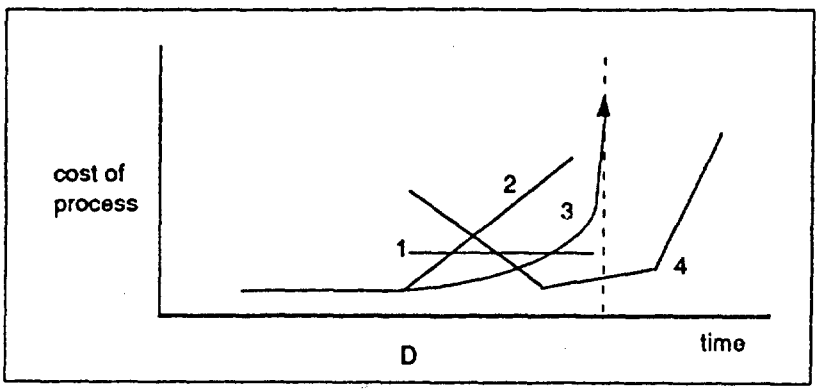


Figure 1. D is known, c(P) is known, P begins at D.

In Figure 1, the process doesn't begin until D, so if we monitor before D, we can't monitor the process, itself. We can, however, monitor a clock. This is the general "I wanna be there when it starts" problem.

Case 1: A one-time cost is incurred at time D. For example, if you don't get the video back to the video store before closing time, a one-time late fee is imposed. You don't have to monitor P, the fining process.

Case 2: A linearly-increasing cost is incurred at time D. For example, if you don't get a book back to the library by a particular date, then an accumulating daily fine is incurred. In this example, the cost doesn't increase continuously, but, rather, as a step function. Accumulating discontinuous functions can look like one-time costs or like continuous functions, depending on the time scale of monitoring. Again, you don't have to monitor P, the fining process.

Case 3: Non-linear cost function. For example, if you don't return a book to your thesis advisor, the cost increases nonlinearly. Again, you don't have to monitor P, the advisor-getting-angry process. In addition, any process might incur infinite cost after some period of time, as illustrated for case 3 by the dashed line in the figure above. For example, at some point, your advisor's dissatisfaction becomes effectively infinite.

Case 4: Nonmonotonic cost function. For example, if D is when cupcakes are put into an oven, then the cost of removing them decreases over the baking period (the cost is high initially because they aren't cooked); and then increases again after the baking period (i.e., they become burned). If you truly know the cost function for P and you know D, then you don't need to monitor P. For example, if cupcakes go in the oven at time D, and you know with certainty that they will be best after 30 minutes, then no monitoring of the cooking process P is necessary; you simply remove them at time D + 30.

So in Cases 1 – 4, monitoring is unnecessary. The basic strategy is to set a timer to go off at time D, or D – 5 for the parking meter example (Case 1), or D + 30 for the cupcakes example. Provided that the timer is accurate, as discussed below.

Implicit in Figure 1 is the idea that we are monitoring to *act at the right time*. This is a fundamental assumption: the value of action depends on timing. In case 1 the right time is before D; in cases 2 and 3, it is as early as possible after D, and in case 4, it is at the minimum point of the cost curve. We assume, then, that there is one or more "best" times or intervals in which to act with respect to any process. But we cannot assume that this time is independent of monitoring actions. For example, if we take the cupcakes out of the oven to monitor their state, then we inadvertently allow the cupcakes and the oven to cool, increasing the time at which they are cooked. We will discuss this issue further, below, in the context of costs of monitoring.

The next four cases parallel the previous four, except D is unknown.

Case 5: P begins at D, but D is unknown. A one-time cost is associated with P. For example, every now and then, you will eat something that makes you sick. You don't know when it will happen so you can't avoid it. You will be out of commission for a day or two. In terms of monitoring, this situation is hopeless. You can only pay the

cost. If you know what affects the probability of P, then you can try to avoid P; and if you know how to keep the costs low, then you may not do too badly; but there's really no way to win in this situation.

Case 6: P begins at D, but D is unknown. A linearly increasing cost is associated with P. For example, fires start in Phoenix with some frequency, but the exact time at which a fire will begin is unknown. A linearly (or perhaps superlinearly) increasing cost is associated with each fire. The best monitoring strategy is to monitor frequently enough to minimize the total cost of fires and monitoring. We can show that this is $\sqrt{2H/Fp}$, where H is the cost of a single monitoring event, F is the cost per unit time of one fire, and p is the probability that a fire will begin within a single time unit (Fatima 1992).

Case 7: Like case 6, except the cost function increases nonlinearly. The ideal monitoring frequency wouldn't be $\sqrt{2H/Fp}$, because it was based on a constant cost per unit time of one fire (F). We should derive the ideal frequency for other, common, nonlinear cost functions. Infections, infestations, and other nasty plagues fall into this category, because their cost grows very quickly.

Case 8: P begins at D, but D is unknown. A nonmonotonic cost is associated with P. Imagine that you want to meet with your advisor, so you go to his office and find him working. If he has just started working, then you won't interrupt him much and he won't be too unhappy; if he is just finishing his work, the same is true. But if he is in the middle of something, then he'll yell at you. We don't know what's the best monitoring strategy here.

Imagine you don't know when a meeting is supposed to begin, so you go to the room and, through the closed door, hear that somebody is giving a speech. You reason that if you barge in now, then you will have to contend with two sources of criticism: disturbing the speaker, and arriving late. If you wait for a while, then the speaker will finish and you will be able to enter during a break, avoiding one or perhaps both sources of criticism.

In cases 1 – 4 you know D precisely, and in cases 5 – 8, D is unpredictable. Between these extremes are those cases in which a process is expected to begin at D but might be a bit early or late, or the process does begin exactly at D, but your clock is a bit inaccurate, or both. The "cupcake study" conducted by Ceci & Bronfenbrenner (1985) provides one good example: The cupcakes go into an oven at time D, and are expected to be perfect at D + 30. In fact, the oven may be a bit hot or cool, so the cupcakes may take little more or less time to reach perfection. Moreover, you decide to monitor not the cupcakes, nor even the clock, but rather, your internal clock. When the internal clock says, "Time's up," you look at the real clock, and if it says "Time's up," you look at the cupcakes. (We return to this problem in Section 3.6.4, where we explain part of the cupcake study's results using an optimal monitoring strategy for this problem.)

There are really no monitoring strategies for cases 1 – 4: you just set a timer for some time relative to D. The strategies for cases 6 – 8 involve *periodic* monitoring because you have *no* information, besides the probability of process P, about when it will begin. Indeed, we have shown that under these conditions *periodic monitoring is optimal* (Atkin 1992a).

But what are the right strategies if you have some information about D, for example, a probability distribution of values of D? what is the right strategy if your clock is a bit inaccurate? The "mature" strategy (i.e., the one adopted by 14-year-olds and not by 10-year-olds in the cupcake study) is to *increase the frequency of monitoring* as the time nears D. We address the question of whether this strategy is optimal below.

We have looked at cases where you have to monitor a process or a clock to see when to act. Let us now look at three cases where the process itself "announces" when it is time to act, or roughly when to act. For example, imagine sitting in a train station awaiting a train that should arrive at time D. Instead of waiting at the track (monitoring), one might rely on the train whistle to announce its arrival. On the other hand, some processes incur costs very, very quickly, so one cannot wait for them to announce themselves. For example, if you wait for the train in the restaurant, you might not be able to run to the track following the whistle before the train leaves. Similarly, if you wait for an infection to "announce itself," it might be too late to cure it.

Case 9: Unlike case 1, where the process doesn't begin until time D, in this case the process begins at some earlier time and you are expected to act on or around time D. The process is itself unmonitorable. For example, a friend says she will arrive for dinner at 7pm after a one-hour drive. Although you can't monitor her progress, you can monitor the time, so you uncork the wine at 6:50 pm. In this example, the cost function for the wine is nonmonotonic: for the first 10 minutes the wine improves, then it is drinkable for a few hours, and then it becomes undrinkable. Other versions of this case can be generated with differently-shaped cost functions.

Case 10: P is itself unmonitorable or doesn't begin before time D, but it *announces* when it is ready for you to act (on or around time D), and the window for action is wide enough to act. Notice that in this case, it isn't necessary to know D—the time relative to which you should act—but it isn't possible to time one's actions *before* D unless one knows D, or unless the window for acting is very wide. For example, if you don't know when your friend will arrive for ice-cream, you shouldn't take the ice cream out of the freezer before she arrives. The window between her arrival and the time she gets impatient for ice-cream is wider than the ten minutes required to prepare dessert. On the other hand, you can uncork the wine (because of the wide window). And, of course, if you know D exactly, you can act in advance of D.

Case 11: P is itself unmonitorable or doesn't begin before time D. P announces when it is ready for you to act, but the window for action is *not* wide enough. For example, the airport staff announce the embarkation of your flight, but you are in a bar and

haven't paid for your drinks yet. In such a case, the shape of the cost function matters. This example is a one-time, very high cost (missing the plane).

Note that cases 9 and 10 each involve coordination of two processes. One begins with the arrival of your friend for dinner or for ice cream; the other begins with your preparations for dinner or ice cream. Imagine that your friend is desperate for ice cream and must have it the moment she arrives. You must weigh two costs: the cost of keeping your friend waiting (monotonically increasing) and the cost associated with timing the removal of the ice cream from the freezer (nonmonotonic). The goal is to have the ice cream ready and waiting when your friend arrives, but as the following figure shows, it may be less expensive to risk your friendship than your ice cream. Assume that your friend is expected to arrive at time zero. You could open your ice cream ten minutes early, in which case its cost will be zero ("perfect" ice cream). Or you can wait to open it until your friend arrives. In the latter case, you have two costs to consider: the cost to your friendship of a delay (denoted W) and the cost of eating the ice cream before it is perfect (denoted E). The sum of these costs is denoted S. As shown in Figure 2, M exceeds S at about 6.5 minutes after the expected arrival time (on the other hand, your friend could eat immediately, so this picture isn't quite what we need to discern the tradeoff). But anyway, if your friend is more than a few minutes late, then the strategy of opening the ice cream prior to her arrival is expensive. On the other hand, the strategy of opening the ice cream when she arrives and waiting until it is perfect isn't optimal, either. Note that although E is minimum (the ice cream is perfect) at 10, S is minimum at about 8, suggesting that a compromise between your friend's hunger and ice cream perfection will most endear you to your friend.

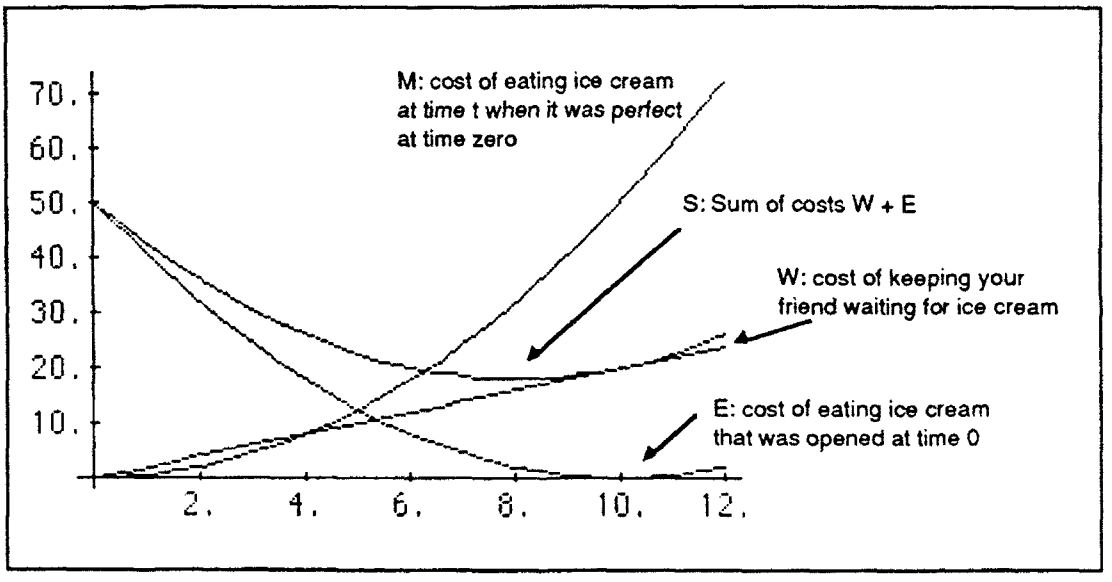


Figure 2. Monitoring Problem Involving Coordination: When to Put Ice Cream in the Freezer?

In cases 1 – 4 we assumed we knew the cost function and D. Thus, if we had to act before D (case 1) we could; if we had to act as soon as possible after D (cases 2 and 3) we could; and if we had to act some time after D that depended on the cost function (e.g., in case 4 we had to act at the minimum value) we could. In all these cases, the process didn't begin until D.

In cases 9 – 11, the process began before D, but the interpretation of D wasn't very different: D is a time relative to which we must act. The best results would be obtained by acting on or around time D. In case 9, we introduced the idea that P might not be observable; but since we know D, and we know we have to act on or around D, we don't have to observe P, it would suffice to observe a clock. We didn't consider the possibility that our clock or our knowledge of D is inaccurate. In case 10, we didn't even have to know D, because P itself "announces" D. We can't act before D, but there's ample time to act once D is announced. In case 11, D is announced, but there isn't time to act. The problem in cases 10 and 11 is that we have to sit around, waiting for D to be announced (case 10), or closely monitoring P for some precursor to D (case 11), when we might be doing more useful work.

3.6.1 A Taxonomy of Monitoring Situations

The similarities between some of these cases suggest that there might be relatively few elementary monitoring situations, and these are combined to form more complex ones. What follows is a first attempt at a taxonomy. Recall that we are interested in timing, that is, when to act. Throughout, we assume there is a time D, and that the value or cost of action depends on when it occurs relative to D. We further assume that action should take place "on or around" D, denoting as D ± i. Then:

Action should take place within a *window*.

- D ± i might be when the window opens
- D ± i might be when the window closes
- D might be somewhere in the window

About windows:

- The window might be open-ended
- The window is not open-ended, but it is wide enough for us to act. However, if we wait to act until we detect that the window opens, the window might close before we can act, depending on how effectively we monitor
- Even if we act the instant the window opens, the window might still close before the action is complete

About D:

- D is known accurately
- D is known, but not accurately
- D is completely unpredictable
- D *announces* itself

About costs:

- The cost function for process P is known
- The general shape of the cost function for P is known (i.e., linearly increasing)
- The cost function for P can be estimated, perhaps from earlier observations
- The cost function for P is unknown
- The cost function for monitoring is known
- The cost function for monitoring is not known
- The cost of monitoring is large/small relative to the cost of P

When processes begin:

- Process P begins at D ± i
- P is ongoing (begins long before D)

Information about process states:

- The state of P can be monitored
- Something correlated with the state of P (e.g., a clock, perhaps not very tightly correlated) can be monitored.
- Nothing about the state of P can be monitored

3.6.2 Monitoring Plan Execution: A Short Survey

In Hansen & Cohen (1992c) we survey the progress that has been made on the problem of monitoring plan execution in the twenty years since the first, simple scheme was developed (Fikes 1971; Fikes, Hart & Nilsson, 1972). Although the research team that built the Shakey robot gave as much attention to the problem of execution monitoring as they did to the problem of plan generation, for more than a decade afterwards the research community focused almost exclusively on the problem of plan generation. The problem of execution monitoring was regarded, at best, as a side-issue. However the last few years have seen a revived interest in plan execution systems, an interest spurred by the desire to build agents that can operate effectively in complex, changing environments. With this in mind, it seems worthwhile to collect together in one place pointers to the scattered work that has been done on this subject, to provide some structure to it, and to analyze the issues it raises. While many good surveys of plan generation have been written, until now no comparable survey has been made of the work that has been done on execution monitoring. (This survey, to appear in *AI Magazine*, is part of Eric Hansen's Master's project.)

As a broad characterization, execution monitoring is a way of dealing with uncertainty about the effect of executing a plan. In general, there are two reasons for monitoring plan execution. The more basic one, which might be called "monitoring for failures", simply consists of checking that a plan works, that actions have their intended effect. The second might be called "monitoring for opportunities". It involves checking for things that need to be done, for events that need to be responded to. It can also involve checking for shortcuts or "optimizations" to a plan that may become available as the plan executes. In this case, the question is not whether a plan works but whether it could be made better.

This survey addresses the following broad questions:

What to Monitor. We begin by considering the question of what to monitor. An answer is to monitor what is relevant, and to determine what is relevant by using the dependency structure of the plan.

When to Monitor. There are two ways in which the question of when and how often to monitor has been addressed. One is by analyzing the uncertainty introduced into a plan by an agent's own actions, and triggering a monitoring action whenever the cumulative uncertainty exceeds some threshold. The other is by modeling the rate of change of the process in the environment being monitored, and setting the monitoring frequency to reflect that rate of change.

Monitoring and Sensing. There is a close relationship between monitoring and sensing, so much so that it can seem natural to identify the two, to say they are one and the same. However in a number of schemes for execution monitoring, monitoring and sensing are distinguished.

Architectures for Monitoring. After investigating the question of what conditions to monitor, as well as when and how often to monitor them, we look at the relationship between monitoring and sensing. The latter question brings us to the point of considering how schemes for monitoring affect, and are affected by, the design of an agent architecture, a subject we could refer to as "how to monitor".

3.6.3 A Phoenix Monitoring Problem (How often to look for new fires)

For the monitoring problem described in Case 6 in Section 3.6 (How often should a helicopter be sent out to detect new fires in Phoenix?), we were able to derive a provably optimal monitoring strategy. The question arises, if such an optimal monitoring strategy exists, do humans use it to address this monitoring problem? We decided to conduct an experiment with human subjects to answer this question.

As in Case 6, let us consider a fire-monitoring task, in which the event to be detected is the start of a fire. In order to determine the optimum monitoring frequency, one needs to know: how frequently fires are set, what is the cost of monitoring, and what is the cost of damage due to the fire. A mathematical model for monitoring frequencies is already available ($\sqrt{2H/Fp}$). We were interested in performing an experiment to check whether humans would converge on optimal monitoring frequencies. If so we could say that the model was true in general. If not we could learn from the deviations from optimality about how humans set monitoring frequencies. The "boxes" experiment, named for the stylized graphics shown to the user during each trial, is described in Fatima (1992).

The boxes experiment was conducted with 30 human subjects and 30 optimum statistical learners, in order to be able to make statistical comparisons of their performances on the monitoring task. From the data collected the cumulative average costs were computed for each of the 30 human subjects and each of the 30 OSLs. It was found that the mean of the cumulative average costs of the 30 subjects was greater

than the mean of the cumulative average costs of the 30 OSLs. We concluded that *human subjects performed suboptimally in this monitoring task!*

We suggested four possible reasons for this, and did further experiments to investigate one of these possibilities, that humans do not monitor optimally because they rely on strategies that are not optimal. We programmed two of the strategies that subjects claimed to use and found that although they performed quite well (in fact better than the human subjects on average) they still performed sub-optimally. This provides some evidence for the conclusion that humans monitor sub-optimally because the strategies they rely on consist of heuristics that give good, but sub-optimal, performance. It would also be interesting to decide what level of performance can be considered to represent deviation from the model. This is because humans may follow the model, but differ from optimal performance due to factors like fatigue, cognitive distraction, and learning. Further work can be done to take these factors into account.

This work was Sameen Fatima's Master's project.

3.6.4 Monitoring to Predict When a Task Will Finish

In this section we consider another of the monitoring situations described in Section 3.6 – the "cupcake study." In their paper, "Don't Forget to Take the Cupcakes out of the Oven: Prospective Memory, Strategic Time-Monitoring, and Context", Ceci and Bronfenbrenner (1985) describe the behavior of children (in two groups, aged 10 and 14) given a simple monitoring task. Cupcakes are placed in an oven to bake, and each child is given the responsibility for taking them out at a predetermined time. While waiting, the child is allowed to play Pac-man. The only way he has of knowing the cupcakes are ready is a clock on a wall behind him, and to see it he has to stop playing Pac-man to look over his shoulder. This setup made it possible for the experimenters to observe how the children monitored the clock.

What they observed was that children monitored according to a "U–shaped distribution" made up of three phases; an "initial flurry of clockchecking activity" followed by a "prolonged period of reduced clock-checking" and then a "relatively rapid burst of last minute clock checks". The authors argue, on intuitive grounds, that this U-shaped distribution represents an efficient monitoring strategy. They explain the first phase as a period of calibration in which frequent clock-checking synchronizes the subject's internal clock with the passage of real time. In the middle phase, the reduced rate of clock-monitoring allows the subject to enjoy the Pac-man game. The "scalloping effect" of the last phase, however, is left unexplained by the authors; it is unclear whether this is because they cannot explain it, or whether they simply consider an explanation too obvious for words. In any event, in this note I want to remedy their omission by explaining why increased clock-checking as a deadline approaches is an efficient strategy for monitoring. Moreover, instead of relying on intuition, I will base the explanation on mathematical reasoning.

Solving this very simple monitoring problem is of little importance in itself, really little more than an exercise. In its simplicity, however, it helps illustrate in a clear way

some of the defining characteristics of monitoring problems. Solving it, therefore, may illustrate an approach to solving monitoring problems in general. At the very least, the solution may give us a technique to add to our tool-kit for solving similar problems in the future.

Monitoring is a way of coping with uncertainty about the state of the world, especially when it may undermine planning. When we plan to do something, we rely on a "model" that lets us predict the effect of our actions and what is likely to happen in the world. We use the word "model", in this context, to refer to a representation of our knowledge about the state of the world, and of cause and effect relationships in it; a model may be a database of predicate calculus statements, a set of differential equations, or may take any of a number of forms. The point is that realistic models are inherently uncertain, and become increasingly unreliable over time unless we periodically monitor the world to update them. Monitoring is a way to synchronize our models with the world.

Designing an efficient monitoring strategy, therefore, means considering questions like; how fast do our models become out of sync with the world, how much does this uncertainty affect the soundness of our decisions, and how often should we monitor to keep this uncertainty within a tolerable limit? Even a monitoring problem as simple as the one described in the Ceci/Bronfenbrenner study raises these questions.

Each child in that study may be said to have an "internal psychological clock" that he uses to estimate how much time has passed. Because this internal clock cannot be relied upon for more than a rough estimate, the child occasionally needs to check a real clock to be sure how much time has actually passed. In terms of the characteristics that I described a monitoring problem as having, the internal clock corresponds to a model, the real clock corresponds to the world, and monitoring the real clock to adjust one's sense of time corresponds to synchronizing the model with the world.

Deciding when to monitor can be viewed, mathematically, as an optimization problem that involves minimizing the combined cost of monitoring (in this case, interrupting the Pac-man game) and the cost of missing the deadline (forgetting to remove the cupcakes from the oven). The risk of missing the deadline is proportional to uncertainty about the time as well as how close to the deadline one is. In Hansen (1992), we show, mathematically, that increased clock-checking as a deadline approaches is an optimal strategy for monitoring.

Mathematical Statement of the Problem. A mathematical framework that can be used for optimization problems of this kind is decision theory. Translating this particular monitoring problem into decision-theoretic terms requires defining a cost function for monitoring, a cost function for missing the deadline, and a probability function to describe the uncertainty of the internal clock. What makes this problem difficult is that a monitoring strategy must consider the possibility of monitoring a repeated number of times before reaching the deadline. A single monitoring decision cannot be analyzed in abstraction from the monitoring decisions that may need to be

made after it. In order to determine when to monitor next in order to minimize cost, we need to know the expected cost of having to monitor again if we fall short of the deadline. This seems to leave any attempt to analyze the problem incomplete.

Solution. The problem seems difficult because we are looking at it in the "forward" direction. By looking at it "backwards", it is possible to see a way to solve it. In order to solve the monitoring problem for time t, we must know the optimal monitoring strategy -and expected cost- for every time less than t. These may be regarded as smaller "subproblems" to be solved before we can solve the original problem. An optimization technique that looks at problems in this way, solving smaller versions of the problem as a way of building a solution to the original problem, is *dynamic programming*. It is an especially good technique for solving problems that involve making a sequence of decisions, sometimes called "multi-stage decision problems". The details of the dynamic programming solution to this monitoring problem appear in Hansen (1992).

Addendum: Proportional Reduction. Our first intuition about this problem was that it could be solved by a simple "proportional reduction" algorithm, which, for any amount of time remaining, say t_r, would schedule the next monitoring event at time $t_r p$ where p is a constant. Thus, if p were .5, each monitoring event would occur after half the remaining time to the deadline had elapsed. Proportional reduction (P.R.) would cause one to monitor more frequently as the deadline approached, the effect seen in the cupcake study. However, we know P.R. is not an optimal strategy for the following reasons:

- For some problems, there is a point at which we should not bother monitoring, either because we are too close to the deadline or because we are too far away. P.R. does not tell us this.
- P.R. is represented graphically by a straight line. But we have shown that an optimal monitoring strategy is represented graphically by a gradual concave curve.
- Most important of all, we don't have a utility model that tells us what proportion to use. Without a utility model, we don't know how to change the proportion when the probability of missing the deadline changes (for example, if the size of the window changes), or when the costs change.

The first two reasons establish that P.R. is not optimal. The third raises the question whether P.R. can even be a useful approximate strategy. Before we can say that it might be, we need a utility model that tells us which proportion to use, and we don't have that yet. Ideally, the utility model should determine the proportion that gives the best approximation to the optimal strategy. Even assuming we had a model to tell us what proportion to use, we would then want to say mathematically how good an approximation it gives. Can we set a bound on the relative error, the absolute error, or the average error we are susceptible to by using the P.R. strategy?

To summarize, before we can argue that P.R. is a useful approximate strategy for monitoring, we need two things:

1. a utility model that tells us what proportion to use
2. a mathematical analysis that tells us how good an approximation P.R. gives to an optimal strategy

We don't have either.

3.6.5 Using a Genetic Algorithm to Monitor Cupcakes

The work described in Atkin (1992b) was motivated by an interest in how to adapt an agent's monitoring activities to its current environment and, as the work described above, attempts to fill in a piece of the monitoring strategy taxonomy by solving the cupcake problem. The research goals were twofold:

1) Could a genetic algorithm adapt its monitoring strategy to different situations?
2) What cupcake monitoring strategy would it come up with and how good is it?

The learning algorithm used is a basic genetic algorithm with fixed population size, mutation and crossover rates. An interesting feature of the algorithm is that it performs *fitness scaling* to prevent unusually good individuals from completely dominating the population and under-average ones from completely disappearing. The code to do this is taken directly from Goldberg (1989, pp. 76–79). Because this was a first attempt at learning monitoring strategies, we didn't want to make the problem too difficult. Therefore, the basic monitoring loop is given to the program – it is then the program's job to determine when and how often to monitor.

The monitoring loop has the parameters the program has to optimize:

- number of times to monitor
- first monitoring time
- function expressing how long to wait to monitor next depending on the time remaining till the deadline.

These parameters are what constitutes an individual in the population. The parameter values determine the fitness of a particular individual.

How good is the proportional reduction strategy? While Hansen (1992) clearly showed that the proportional reduction strategy is not optimal in a mathematical sense (see above), it nevertheless seems to be a very good approximation. In summary, we would say that while a non-linear effect seems likely, it is not at all clear from results alone what term in the optimal model describes it the best or how dominant it is.

This work is the subject of Marc Atkin's Master's project.

3.6.6 Monitoring As A Sequential Decision Problem

A system responsible for monitoring a process, state, or condition over time usually monitors it periodically at some fixed rate. Such systems include: closed-loop control systems with a fixed sampling period, plans with decision points at fixed intervals,

sensory systems that monitor periodically to update a world model. Agents that react to a world state that is sensed at a periodic interval. Given the decision to monitor periodically, the only problem is to determine an optimal monitoring rate or sampling period. This requires weighing the cost of monitoring against the need to minimize response delay in order to determine an optimal tradeoff.

Hansen & Cohen (1992b) presents two representative examples in which periodic monitoring is not optimal; both are important in real-time computing. They are: monitoring a task to predict when it will finish, and monitoring a task to predict whether it will meet its deadline. It is shown that in each case the problem of when to monitor is a sequential decision problem. Given a model of the state transition probabilities and payoffs, an optimal monitoring policy can be determined using stochastic dynamic programming. The optimal monitoring policy for each example is characterized and in each case it turns out to be nonperiodic. For the first example, one should monitor more frequently as the task nears completion. This is the task presented above in Section 3.6.4 – the cupcake problem. For the second, one should monitor more frequently as one comes closer to triggering the threshold of the decision rule.

Monitoring to Predict Whether a Task Will Meet a Deadline. For our second example we consider the problem of checking a decision rule that predicts whether a task will finish by a deadline. One instance of this is envelopes, and is implemented as part of our Phoenix planner (Hart, Anderson & Cohen, 1990; also see Section 3.5). The idea of using a decision rule such as an envelope to anticipate failure to meet a deadline soon enough ahead of time to initiate recovery has wide application, especially for real-time computing. AI systems that do approximate processing under time pressure monitor progress so that they can adjust their processing strategy to make sure they generate at least an approximate solution by a deadline (Lesser, Pavlin & Durfee, 1988). Similarly, dynamic schedulers for real-time operating systems monitor task execution so that they can anticipate failure to meet a task deadline as soon as possible and take appropriate action (Haben & Shin, 1990).

Given a decision rule (such as an envelope) that predicts whether or not a deadline will be met, it remains to be decided how often this rule should be tested (in other words, how often should the envelope be monitored). If monitoring has no cost, it can tested continuously. But if it has a cost, there must be a scheduling policy for it. When we built the Phoenix planner we assumed a periodic strategy for monitoring. This was purely ad hoc; we monitored the envelope to check performance every fifteen minutes of simulated time, without regard for the cost of making each check. This is also true in Haben & Shin's dynamic scheduling system, which assumes there is no cost of monitoring.

To solve what we will call the "envelope monitoring problem" using stochastic dynamic programming, we first express it in formal mathematical terms. Its state representation is a bit more complicated than for the cupcake monitoring problem; instead of a single state variable, the state is represented by a vector of two variables: 1)

the time remaining before the deadline, and 2) the distance remaining to reach the goal condition. Again there is a single decision variable, m; the decision is to either stop and abandon the task or to continue executing it for m additional units of time before monitoring its progress again.

The complete solution is given in Hanson & Cohen (1992a). The obvious qualitative observation to make is that the frequency of monitoring increases with closeness to the envelope boundary. This supports the intuition that the more likely one is to cross the threshold of a decision rule, the more often one should check (or "monitor") the rule.

We are currently working to reconcile this optimal but costly strategy (time complexity $O(n^3)$ and space complexity $O(n^2)$) with the one-parameter slack-time envelope boundary estimator discussed in Section 3.5. Our intuition is that the one-parameter rule can serve as a cheap approximation of the optimal strategy at times when there is insufficient time to compute it.

3.6.7 Learning a Decision Rule for Monitoring Tasks with Deadlines

In the preceding section (and in Hanson & Cohen 1992b) we showed that, given a model of the state transition probabilities and payoffs, an optimal monitoring policy can be determined using stochastic dynamic programming. In Hanson and Cohen (1992c) we extend this work, showing that even without a model, an optimal monitoring policy can be *learned*.

A real-time scheduler or planner is responsible for managing tasks with deadlines. When the time required to execute a task is uncertain, it may be useful to monitor the task to predict whether it will meet its deadline; this provides an opportunity to make adjustments or else to abandon a task that can't succeed. Hanson & Cohen (1992b) treats monitoring a task with a deadline as a sequential decision problem. Given an explicit model of task execution time, execution cost, and payoff for meeting a deadline, an optimal decision rule for monitoring the task can be constructed using stochastic dynamic programming. If a model is not available, the same rule can be learned using *temporal difference methods* (Barto, Sutton & Watkins, 1990). These results are significant because of the importance of this decision rule in real-time computing.

It makes sense to construct a decision rule such as the one described in Hanson & Cohen (1992b) for tasks that are repeated many times or for a class of tasks with the same behavior. This allows the rule to be learned, if TD methods are relied on; or for statistics to be gathered to characterize a probability and cost model, if dynamic programming is relied on. However if a model is known beforehand, or can be estimated, a decision rule can also be constructed for a task that executes only once.

The time complexity of the dynamic programming algorithm is $O(n^2)$, where n is the number of time steps from the start of the task to its deadline; however the decision rule may be compiled once and reused for subsequent tasks. The time complexity of

TD learning, $O(n)$, is mitigated by the possibility of turning learning off and on. The space overhead of representing an evaluation function by a table is avoidable by using a more compact function representation, such as a connectionist network.

Besides the fact that this approach is not computationally intensive, it has other advantages. It is conceptually simple. The decision rule it constructs is optimal, or converges to the optimal in the case of TD learning. It works no matter what probability model characterizes the execution time of a task and no matter what cost model applies, and so is extremely general. Finally, it works even when no model of the state transition probabilities and costs is available, although a model can be taken advantage of.

These results can be extended in a couple obvious ways. The first is to factor in a cost for monitoring. Our analysis thus far has assumed that monitoring has no cost, or its cost is negligible. This allows monitoring to be nearly continuous, in effect, for a task to be monitored each time step. Others who have developed similar decision rules have also assumed the cost of monitoring is negligible. However in some cases the cost of monitoring may be significant, so in another paper we show how this cost can be factored in (Hansen & Cohen 1992b). Once again we use dynamic programming and TD methods to develop optimal monitoring strategies.

The second way in which this work can be extended is to make the decision rule more complicated. Here we analyzed a simple example in which the only alternative to continuing a task is to abandon it. But recovery options may be available as well. A dynamic scheduler for a real-time operating system is unlikely to have recovery options available, but an AI planner or problem-solver is almost certain to have them (Lesser, Pavlin & Durfee, 1988; Howe, 1992). The way to handle the more complicated decision problem this poses is to regard each recovery option as a separate task characterized by its own probability model and cost model; so at any point the expected value of the option can be computed. Then instead of choosing between two options, either continuing a task or abandoning it, the choice includes the recovery options as well. The rule is simply to choose the option with the highest expected value.

The work described in this and the preceding subsection constitute the second half of Eric Hansen's Master's project.

4 Publications, Technical Reports, Presentations and Honors

Refereed Papers Published

Anderson, S.D., Hart, D.M., & Cohen, P.R., 1991. Two ways to act. *AAAI Spring Symposium on Integrated Intelligent Architectures*. Published in the *SIGART Bulletin*, Vol. 2, No.4, pp. 20-24.

Cohen, P.R & Hart, D.M., 1993. Path analysis models of an autonomous agent in a complex environment. To appear in *Proceedings of the Fourth International Workshop on AI and Statistics*, Ft. Lauderale, FL, January 1993.

Cohen, P.R., St. Amant, R. & Hart, D.M., 1992. Early warning of plan failure, false positives and envelopes: Experiments and a model. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates, Inc. Pp. 773-778.

Garvey, A. & Lesser, V., 1992. Design-to-time real-time scheduling. To appear in *IEEE Transactions on Systems, Man and Cybernetics*.

Hart, D.M. & Cohen, P.R., 1992. Predicting and explaining success and task duration in the Phoenix planner. *Proceedings of the First International Conference on AI Planning Systems*. Morgan Kaufmann. Pp. 106-115.

Refereed Papers Submitted

Hanks, S., Pollack, M.E. & Cohen, P.R., 1992. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. To appear in *AI Magazine*.

Hansen, E.A. & Cohen, P.R. 1992. Learning a decision rule for monitoring tasks with deadlines. Submitted to the *Thirteenth International Joint Conference on Artificial Intelligence*. Also, Computer Science Technical Report 92–80. Univ. of Massachusetts. Amherst.

Hansen, E.A. & Cohen, P.R. 1992. Monitoring as a sequential decision problem. In preparation for the *Eleventh National Conference on Artificial Intelligence*.

Hansen, E.A. & Cohen, P.R. 1992. Monitoring plan execution: A survey. In preparation for *AI Magazine*.

Unrefereed Reports and Articles

Anderson, S.D. & Hart, D.M.. 1990. Monitoring interval. *EKSL Memo #11*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Atkin, M., 1992. Research summary: Using a genetic algorithm to monitor cupcakes. *EKSL Memo #24*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Atkin, M., 1992. A note on periodic monitoring. *EKSL Memo #23*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Cohen, P.R., 1991. Timing is everything. *EKSL Memo #21*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Cohen, Paul R., Anderson, Scott D., Hart, David M., 1990. Scheduling Agent Actions in Real-Time. Abstract for the *Interdisciplinary Workshop on the Design Principles for Real-Time Knowledge Based Control Systems* at the *Eighth National Conference on Artificial Intelligence*. Boston, MA.

Fatima, S.S., 1992. The boxes experiment: Learning an optimal monitoring interval. Computer Science Technical Report 92-38. Univ. of Massachusetts. Amherst.

Hansen, E.A., 1992. Note on monitoring cupcakes. *EKSL Memo #22*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Hansen, E.A., 1990. The effect of wind on fire spread. *EKSL Memo #10*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Hansen, E.A., 1990. A model for wind in Phoenix. *EKSL Memo #12*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Hart, D.M. & Cohen, P.R., 1991. Phoenix: A testbed for shared planning research. *Collected Notes from The Benchmarks and Metrics Workshop*, sponsored by NASA and DARPA at NASA Ames.

Silvey, P.E., 1990. Phoenix baseline fire spread models. *EKSL Memo #13*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Books or Parts Thereof

Cohen, P.R. *Empirical Methods in Artificial Intelligence*. Textbook in preparation.

Cohen, P.R., 1990. Architectures for reasoning under uncertainty. *Readings in Uncertain Reasoning*. Glenn Shafer and Judea Pearl (Eds). Morgan Kaufmann.

Invited Presentations

Cohen, P.R.

- Member of a panel entitled "Planning under uncertainty" at the *AAAI Workshop on Production Planning, Scheduling and Control*, which focused on scheduling strategies for managing uncertainty in complex, real-time environments, July 1992.

- Invited presentation on EKSL's current research at a two day meeting of the Institute for Defense Analysis's Information and Science Technology Advisory Group on Simulation in Washington, D.C., June 17-18, 1992.
- Member of a panel entitled "The empirical evaluation of planning systems: Promises and pitfalls" at the *First International Conference on AI Planning Systems* at the Univ. of Maryland, June 16, 1992.
- Methods for agentology: General concerns, specific examples. Invited talks at Virginia Polytechnic Institute and the Univ. of West Virginia. April 1992.
- Three examples of statistical modeling of an AI program. Invited talk at the Univ. of Texas, Austin. March 1992.
- Member of a panel entitled "The future of expert systems" chaired by Dr. Y.T. Chen of NSF at the World Congress on Expert Systems, December 16-19, in Orlando, Florida.
- A brief report on a survey of AAAI-90, some methodological conclusions, and an example of the MAD methodology in Phoenix. Keynote address, *AAAI Spring Symposium on Implemented AI Systems*. Palo Alto, CA. March, 1991.
- What is an interesting environment for AI planning research? Panel moderator, *Workshop on Automated Planning for Complex Domains* at the *Eighth National Conference on Artificial Intelligence*. Boston, MA. July 30, 1990.
- Methodological problems, a model-based design and analysis methodology, and an example. Keynote address at the *International Symposium on Methodologies for Intelligent Systems*. Knoxville, TN. October 25-27, 1990.

Contributed Presentations

Cohen, P.R.

- The Phoenix project: Responding to environmental change. *Workshop on Innovative Approaches to Planning, Scheduling, and Control* . San Diego, CA. November 1990.

Hart, D.M.

- Predicting and explaining success and task duration in the phoenix planner. Paper presentation at the *First International Conference on AI Planning Systems* at the University of Maryland, June 15, 1992.

Howe, A.E.

- Designing agents to plan and act in their environments. *Workshop on Automated Planning for Complex Domains* at the *Eighth National Conference on Artificial Intelligence*. Boston, MA. July 30, 1990.

Tutorials

Cohen, P.R.

- Offered a tutorial with Bruce Porter (of the University of Texas) at the *Tenth National Conference on Artificial Intelligence* entitled "Experimental Methods in Artificial Intelligence." This tutorial used several examples from our research

under this contract, including the results reported in (Hart & Cohen 1992) and (Cohen, St. Amant & Hart, 1992).

Hart, D.M.

- Tutorial on the Phoenix Testbed and IRTPS research being conducted in Phoenix at the IRTPS contractors' meeting, Wright Patterson AFB, April, 1991. This tutorial was offered for potential consumers of IRTPS research results.

Honors, including conference committees

Cohen, P.R.

- Elected a Councilor of the *American Association for Artificial Intelligence* for the term 1991-94.
- Appointed to the Information and Science Technology Advisory Group on Simulation, Institute for Defense Analysis.
- As a AAAI Councillor, serving as Chair of the AAAI-93 Tutorial Committee, Co-chair of the 1992-93 Symposium Committee, and Assistant to the Chair for the Program Committee of AAAI-93.
- Chairman, NSF/DARPA *Workshop on AI Methodology*. University of Massachusetts. June, 1991
- Organizing Committee, *AAAI Workshop on Intelligent Real-Time Problem Solving*. Anaheim, CA. July, 1991.
- Program Committee, *Sixth International Symposium on Methodologies for Intelligent Systems* (ISMIS'91). Charlotte, NC. October 1991.

5 List of Professional Personnel

Professor **Paul R. Cohen**, principle investigator, guided most of the research reported on here, including the work on baselining, envelopes, and monitoring strategies.

Professor **Victor R. Lesser**, principle investigator, directed the research into real-time scheduling of cognitive tasks, described in (Garvey & Lesser, 1992).

David M. Hart, lab manager, supervised many of the research activities reported here. Hart coordinated the activities of research assistants working on envelopes and monitoring in Phoenix. He also analyzed results of the Phoenix real-time baselining experiment, working with Cohen to investigate techniques for modeling these results. This search led to the recognition of path analysis as a powerful tool for building causal models from empirical data (Hart & Cohen, 1992).

David L. Westbrook, systems developer, implemented instrumentation and facilities to support baseline scenarios and experimentation and co-authored the Phoenix testbed documentation. Westbrook is responsible for maintenance of Phoenix and the associated testbed environment. In addition, Westbrook helped design and implement the following experiments:

- real-time baselining experiment (Hart & Cohen, 1992)
- slack-time envelopes experiment (Cohen, St. Amant & Hart, 1992)
- interactive experiment designed to measure optimality of human monitoring strategies (Fatima 1992).

6 Interactions

DARPA/Rome Labs Planning Initiative

Much of the work we have done for the IRTPS initiative is being transferred to the DARPA/Rome Labs Planning Initiative (PI). This includes the creation of a testbed environment for controlled experimentation (our Phase I work), our ongoing work with envelopes and monitoring, and the use of path analysis to build causal models of program behavior. Part of the PI effort involves building a Common Prototyping Environment (CPE) for integrating and evaluating components of the evolving planning and scheduling architecture. The CPE will have many of the kinds of testbed features we built into Phoenix and are building into our simulation of the transportation planning domain.

Using Simulation Testbeds to Design AI Planners. During Phase I of the IRTPS Initiative we created a testbed environment for Phoenix. This effort included instrumenting the system, baselining the simulated environment, and providing such facilities as predefined scenarios, scripts, and primitives for experiment definition and data collection. We integrated the first version of CLASP[3] into this testbed environment to provide built-in data analysis capabilities. Since that time we have extended many of these testbed features in Phoenix and ported them to our simulation of the sea transport domain in the PI. Many of these features will soon find their way into the CPE under the direction of the Issues Working Group on Prototyping Environment, Instrumentation and Methodology. Paul Cohen is the co-chair of this group along with Mark Burstein of BBN.

Envelopes and Monitoring. Our work with envelopes and monitoring under Phase III is directly applicable to the design of *pathology demons* for the transportation planning problem that is the domain of the PI. Pathology demons are designed to detect typical pathologies that arise during the execution of large-scale transportation plans and to help the user (interacting through informed visualizations) steer the plan around the pathologies. Envelope-like representations of plan progress tell us whether we are keeping trim to the schedule, and our developing theories of appropriate monitoring strategies tell us how often to monitor and what to watch.

Building Causal Models of Program Behavior using Path Analysis. During Phase III we began applying path analysis to our work in Phoenix (see Section 4). We think path analysis will provide a powerful technique for building causal models from large data sets such as those generated by experiments in Phoenix and the PI's CPE. We recently enhanced CLASP by adding a module for path analysis. Using a graphical interface, the user draws a directed graph of hypothesized causal influences among independent and dependent variables, and the path analysis module calculates the corresponding path coefficients (strengths of influence) along the arcs (one per arc).

---

[3] The Common Lisp Analytical Statistics Package (CLASP) was originally implemented on the TI Explorer for analyzing Phoenix experiments. For more on CLASP, see Fisher (1990).

The user can explore variations on the model simply by modifying nodes and arcs in the graph – recalculation is done automatically. Such a facility will help users fit causal models to planner behaviors that arise in CPE simulations. As part of a new contract in the PI, we will be building an experiment module that automatically generates causal models from data sets to aid developers in the design and evaluation of AI planning systems.

References

Anderson, S.D., 1992. Temporal Projection of Stochastic Process Models. Thesis Proposal, Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Anderson, S.D., Hart, D.M., & Cohen, P.R., 1991. Two ways to act. *AAAI Spring Symposium on Integrated Intelligent Architectures*. Published in the *SIGART Bulletin*, Vol. 2, No.4, Pp. 20-24.

Anderson, S.D. & Hart, D.M., 1990. Monitoring interval. *EKSL Memo #11*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Asher, H.B. 1983. *Causal Modeling*. Sage Publications.

Atkin, M., 1992a. A note on periodic monitoring. *EKSL Memo #23*. Experimenta' Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Atkin, M., 1992b. Research summary: Using a genetic algorithm to monitor cupcakes. *EKSL Memo #23*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Barto, A.G.; Sutton, RS.; and Watkins, C.J.C.H., 1990. Learning and sequential decision making. In *Learning and Computational Neuroscience: Foundations of Adaptive Networks*.. M. Gabriel and J. W. Moore (Eds.), MIT Press, Cambridge, MA. Pp. 539-602.

Bonissone,P.P. & Halverson, P.C., 1990. Time-constrained reasoning under uncertainty. *The Journal of Real-time Systems*, Vol. 2, Nos. 1/2, Pp. 25-45, 1990.

Ceci, S.J. & Bronfenbrenner, U., 1985. "Don't forget to take the cupcakes out of the oven": Prospective memory, strategic time-monitoring, and context. *Child Development*, Vol. 56, Pp. 152-164.

Cohen, P.R. *Empirical Methods in Artificial Intelligence*. Textbook in preparation.

Cohen, P.R & Hart, D.M., 1993. Path analysis models of an autonomous agent in a complex environment. To appear in *Proceedings of the Fourth International Workshop on AI and Statistics*, Ft. Lauderale, FL, January 1993.

Cohen, P.R., St. Amant, R. & Hart, D.M., 1992. Early warning of plan failure, false positives and envelopes: Experiments and a model. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates, Inc. Pp. 773-778.

Cohen, P.R., 1991a. Timing is everything. *EKSL Memo #21*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Cohen, P.R., 1991b. A survey of the *Eighth National Conference on Artificial Intelligence*: Pulling together or pulling apart? *AI Magazine* 12(1): 16-41.

Cohen, P.R., Greenberg, M.L., Hart, D.M. & Howe, A.E., 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3): 32-48.

D'Ambrosio, B., 1989. Resource-bounded agents in an uncertain world. *Proceedings of the Workshop on Real-Time AI Problems*, IJCAI-89. Detroit, MI.

Fatima, S.S., 1992. The boxes experiment: Learning an optimal monitoring interval. Computer Science Technical Report 92-38. Univ. of Massachusetts. Amherst.

Fikes, R., Hart, P. & Nilsson, N., 1972. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4), Pp. 251-288.

Fikes, R., 1971. Monitored execution of robot plans produced by STRIPS. *Proceedings of the IFIP Congress, 1971*, Ljubljana, Yugoslavia. Pp. 189-194.

Fisher, D.E. 1990. Common Lisp Analytical Statistics Package (CLASP): User Manual. Computer Science Technical Report 90–85, Univ. of Massachusetts, Amherst.

Garvey, A. & Lesser, V.R., 1992. Design-to-time real-time scheduling. To appear in *IEEE Transactions on Systems, Man and Cybernetics*.

Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley.

Greenberg, M.L. & Westbrook, D.L., 1990. The Phoenix testbed. Computer Science Technical Report #90-19, Univ. of Massachusetts, Amherst.

Haben, D. & Shin, K., 1990. Application of real-time monitoring to scheduling tasks with random execution times. *IEEE Transactions on Software Engineering*, Vol. 16, No. 12, Pp. 1374-1389.

Hansen, E.A. & Cohen, P.R. 1992a. Learning a decision rule for monitoring tasks with deadlines. Submitted to the *Thirteenth International Joint Conference on Artificial Intelligence*. Also, Computer Science Technical Report 92–80. Univ. of Massachusetts. Amherst.

Hansen, E.A. & Cohen, P.R. 1992b. Monitoring as a sequential decision problem. In preparation for the *Eleventh National Conference on Artificial Intelligence*.

Hansen, E.A. & Cohen, P.R. 1992c. Monitoring plan execution: A survey. In preparation for *AI Magazine*.

Hansen, E.A., 1992. Note on monitoring cupcakes. *EKSL Memo #22*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.

Hart, D.M. & Cohen, P.R., 1992. Predicting and explaining success and task duration in the Phoenix planner. *Proceedings of the First International Conference on AI Planning Systems*. Morgan Kaufmann. Pp. 106-115.

Hart, D.M. & Cohen, P.R., 1991. Phoenix: A testbed for shared planning research. *Collected Notes from The Benchmarks and Metrics Workshop*, sponsored by NASA and DARPA at NASA Ames.

Hart, D.M., Anderson, S.D., & Cohen, P.R., 1990. Envelopes as a vehicle for improving the efficiency of plan execution. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Katia Sycara (Ed.). Morgan-Kaufman. Pp. 71-76.

Howe, A.E., 1992. Analyzing failure recovery to improve planner design. *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI Press/MIT Press. Pp. 387-392.

Howe, Adele E., Hart, David M., Cohen, Paul R., 1990. Addressing real-time constraints in the design of autonomous agents. *The Journal of Real-Time Systems*, Vol. 2, Nos. 1/2, Pp. 81-97.

Lesser, V.R., Pavlin, J. & Durfee, E., 1988. Approximate processing in real-time problem solving. *AI Magazine,* 9(2): 49-61.

Li, C.C., 1975. *Path Analysis–A Primer*. Boxwood Press.

Russell, S.J. & Zilberstein, S., 1991. Composing real-time systems. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*. Sydney, Australia.